Linear Path Skyline Computation in Bicriteria Networks

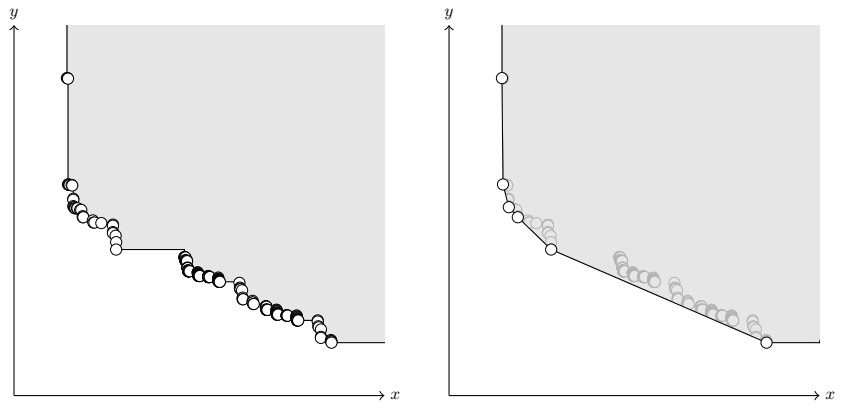
Michael Shekelyan, Gregor Jossé, Matthias Schubert, and Hans-Peter Kriegel

Institute for Informatics, Ludwig-Maximilians-Universität München, Oettingenstr. 67, D-80538 Munich, Germany
{shekelyan,josse,schubert,kriegel}@dbs.ifi.lmu.de

Abstract. A bicriteria network is an interlinked data set where edges are labeled with two cost attributes. An example is a road network where edges represent road segments being labeled with traversal time and energy consumption. To measure the proximity of two nodes in network data, the common method is to compute a cost optimal path between the nodes. In a bicriteria network, there often is no unique path being optimal w.r.t. both types of cost. Instead, a path skyline describes the set of non-dominated paths that are optimal under varying preference functions. In this paper, we examine the subset of the path skyline which is optimal under the most common type of preference function, the weighted sum. We will examine characteristics of this more strict domination relation. Furthermore, we introduce techniques to efficiently maintain the set of linearly non-dominated paths. Finally, we will introduce a new algorithm to compute all linearly non-dominated paths denoted as linear path skyline. In our experimental evaluation, we will compare our new approach to other methods for computing the linear skyline and efficient approaches to compute path skylines.

1 Introduction

In recent years graph data has gained much importance in numerous information management systems. For example, spatial databases no longer rely on free movement and simple distance measures but more restrictive movement patterns, e.g. along streets or railways being modelled as a network. Other applications for graph data include social networks, computer networks or the world wide web itself. In all these networks it is often of interest to determine cost-optimal paths between nodes in order to determine connections or to simply measure the distance. The cost for traversing an edge usually depends on the application, e.g. the number of common friends in social networks or the energy consumption within a street network. Using a single cost criterion is often strongly restrictive. For example, when planning a bicycle route, minimizing only the distance is usually not sufficient. To determine the difficulty of a route, considering the ascension of a path is essential. In order to employ multiple criteria when computing optimal paths, a straight forward way is to combine the cost criteria into a single value and optimize the combined costs. However, finding a



(a) Ordinary Skyline with 190 elements (b) Linear Skyline with 6 elements

Fig. 1. The Linear Skyline consists of a subset of the elements of the Ordinary Skyline.

meaningful cost combination means weighting the impact of each cost criterion. This usually depends on personal preferences, the semantics of the application as well as the scale of the cost criteria. Hence, finding an appropriate weighting is often very difficult. An alternative way to cope with multiple cost criteria is to determine all results that could potentially be optimal under an arbitrary cost combination. In the database community this type of query is known as skyline query [1]. The result set of a skyline query contains elements which are not dominated in the following sense: Object o is dominated by object q if for each cost criterion the cost of o is equal or larger than the cost of q, and for at least one criterion the cost of q is strictly smaller than the cost of o. This means, there is no non-negative cost combination where p would be preferable to q. The task of computing a skyline of paths in a network is called Route Skyline Computation [2] or Multiobjective Shortest Path Search [**?**].

Though posing a skyline query does not require the explicit specification of attribute weights, it still yields two practical problems limiting its usability. The number of non-dominated solutions typically grows superlinear in the number of criteria. In consequence, there often is an abundance of results having very similar costs. Furthermore, for more than two cost criteria, it is hard to determine the conditions and the combination functions which would lead to the optimality of a particular element of the skyline. In order to keep the result set easier to interpret and the amount of potentially optimal paths on a moderate level, we will focus on the basic case of optimizing two cost criteria. This case also exhibits mathematical benefits.

To reduce the amount of results to a more intuitive set, we propose the novel concept of linear skyline queries. A linear skyline consists of the subset of the conventional skyline which is optimal under all linear combination functions. This concept is introduced more formally in Section 3 and visualized in

Figure 1. The linear skyline contains fewer and more diverse results. Also, the characteristics of both skyline notions differ greatly. In contrast to conventional domination, linear domination is defined through sets of objects. In the course of this paper, we will introduce and discuss the concept and the computation of linear skylines.

Though the task of computing a linear skyline itself is novel, there exist approaches which employ the concept as a preprocessing step towards computing conventional skylines. In contrast to existing approaches (cf. Section 2), we propose a two-step method which is based on the idea of label correction for computing the conventional skyline. Our novel method is mainly based on new techniques to efficiently manage linearly non-dominated solutions, allowing efficient updates and deletions. To summarize, the contributions of the paper are the following: the introduction of linear skyline computation as a novel type of query within a network, a discussion of the properties of linear skylines, and an efficient algorithm that exploits the more restrictive dominance relation.

The rest of the paper is organized as follows: In Section 2, we discuss related work. The general setting and formalizations are given in Section 3. Section 4 describes our new algorithm for computing the linear path skyline. In Section 5, we compare our new algorithm to methods that can be adapted to compute linear skylines. Section 6 summarizes the paper and presents some ideas for future work.

2 Related Work

In the database community, the skyline operator was introduced in [1]. Multiple approaches to compute skylines in database systems on sets of cost vectors followed [4–6]. Also, different notions of dominance have been presented. In [7] k-dominant skylines are proposed which generalize the dominance relationship by requiring that an object needs to improve any other object in at least k attributes. The approach in [8] is based on identifying subspaces in which the skyline contains a limited amount of solutions. [9] demonstrates that the selection of the k most representative skyline objects is a non-trivial task and proposes a dynamic programming algorithm for two cost criteria and a polynomial time algorithm for higher dimensionalities. In [10], the authors propose to group skyline points w.r.t. the subspace for which they are part of the skyline.

For network data, the following approaches have been introduced. In [12], the authors introduce a method for calculating a skyline of landmarks in a road network that are compared w.r.t. their network distance to several query objects. In [13] the authors propose in-route skyline processing in road networks. Assume that a user is moving along a predefined route to a known destination, the algorithm processes minimal detours to sets of landmarks being distributed along the path. In [14] the authors discuss continuous skyline queries in road networks, i.e. a moving user queries for a skyline of point of interest. The most similar setting to the approach being presented in this paper is computing route skylines [2]. In this task, we want to find all paths between two nodes in a mul-

ticost network where the costs of paths are not dominated by any other path between the same two nodes. In Operations Research the task of computing a set of non-dominated solutions is known as multiobjective optimization or pareto optimization [**?**]. In case there are only two cost criteria the task is named bicriteria shortest path problem. This setting is of special interest because the amount of results is usually less extensive than in higher dimensions, and the result is much easier to interpret by users. Furthermore, the multicriteria and bicriteria shortest path problems are known to be NP-hard [16]. Thus, the limitation helps to keep the computational effort at an applicable level. Finally, the limitation to a two-dimensional cost space allows several optimizations which cannot be transferred to higher dimensional cost spaces.

In [17] the authors describe a comparison of several state-of-the-art methods. The label correcting method [18] maintains a set of non-dominated solutions at each node. Another approach is the near-shortest path method [20] which is based on the idea of computing all paths having a length within a certain deviation from the length of the optimal path for one criterion. Finally, [21, 17] finds all non-dominated paths in a two-phase approach. The first phase computes the set of so-called supported solutions, followed by the second phase determining the remaining results. These supported solutions are equivalent to the linearly non-dominated solutions we aim to find in this paper. In [17], the authors compared several algorithms for determining the supported solutions. The most successful method for computing k supported solutions is based on $2 \cdot k - 1$ Dijkstra searches, each with a different linear cost combination as optimization criterion. We will compare two improved versions of this approach in our experiments.

Our method does not employ any precomputation step [22] and thus, it is applicable to dynamically changing network costs. To guide our search towards the target, our method generates optimal lower bound estimations as part of the query processing algorithm as proposed in [3].

The linear path skyline as described in this paper is part of of the convex hull over the cost vectors of all paths between start and target. However, the set of all paths is not available at query time. Thus, efficient methods to determine the convex hull cannot be applied to solve the problem being described in this paper.

3 Preliminaries

A bicriteria network is a directed graph $G = (V, E)$ where V denotes the set of vertices and $E \in V \times V$ denotes the set of directed edges. Furthermore, we assume the existence of two cost functions $c_1, c_2 : E \to \mathbb{R}_{\geq 0}$. In case G describes a road network the cost functions may represent height differences, travel distances or the number of traffic lights. Every edge $(u, v) \in E$ is labeled with a non-negative two-dimensional cost vector, $c((u, v)) := ((u, v)_1, (u, v)_2)$. A path or route P is a consecutive set of edges $((s, v), \ldots, (u, t))$ which does not visit any node twice. Likewise, any path has a cost vector $p = (p_1, p_2) = \sum_{(u,v)\in P} c((u, v))$. Throughout this paper, we shall denote paths with capital letters (e.g. P) and

the corresponding cost vector with the same lowercase letter (e.g. p). For multiple paths, we either choose different capital letters or superscript their indices.

Given a start node $s \in V$ and a target node $t \in V$, our algorithm computes a subset of all the routes $\mathcal{R} := \mathcal{R}(s,t)$ starting at s and ending at t. Let us note that the following definitions and theoretical results are equally applicable to any set of vectors in $\mathbb{R}^d_{\geq 0}$ as well.

In order to distinguish between linear skylines and skylines in the ordinary sense, we first recall the definiton of conventional skylines and the conventional dominance relation they are based on. For reasons of generality, we do not restrict our definitions to the bicriteria case.

Definition 1 *Conventional Route Skyline*
Let $\mathcal{R}$ be a set of paths in a d-dimensional cost space. Then $P \in \mathcal{R}$ dominates *$Q \in \mathcal{R}$, denoted as $p \prec q$ ($or P \prec Q$), iff*

$$\exists\ 1 \leq i \leq d : p_i < q_i \wedge \nexists\ 1 \leq i \leq d : p_i > q_i.$$

The set of non-dominated routes, i.e. $\{P \in \mathcal{R} \mid \nexists Q \in \mathcal{R} : q \prec p\}$, is denoted conventional skyline.
Let § be a set of non-dominated paths and Q be a route. If there exists $P \in$ § such that $p \prec q$, we say Q is dominated by §. Conversely, we say Q is non-dominated by § if no such P exists.

After describing conventional dominance, we now present the definintion of linear dominance which is more strict.

Definition 2 *Linear Dominance*
Let $\mathcal{R}$ be a set of paths in a d-dimensional cost space. $P \in \mathcal{R}$ w-dominates *$Q \in \mathcal{R}$, where $0 \neq w \in \mathbb{R}^d_{\geq 0}$, iff $w^T p < w^T q$. w is called a* weight vector.

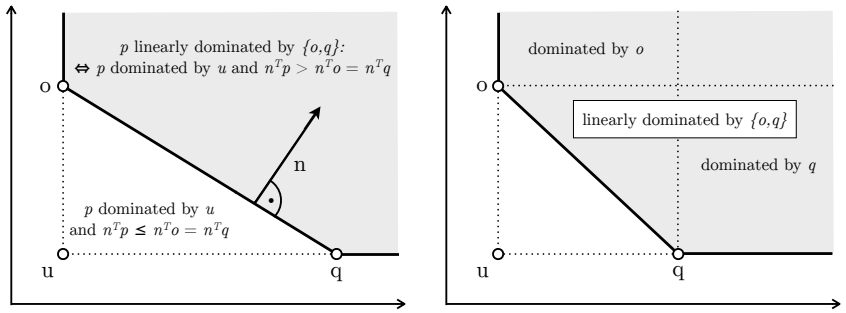
Definition 3 *Linear Skyline*
Let $\mathcal{R}$ be a set of paths in a d-dimensional cost space.
A subset $\mathcal{S}' = \{P^1, \ldots, P^K\} \subseteq \mathcal{R}$ linearly dominates *a path $Q \in \mathcal{R}$, denoted as $\mathcal{S}' \prec_{lin} Q$ ($or\ \{p^1, \ldots, p^K\} \prec_{lin} q$) iff*

$$\exists\ P \in \mathcal{S}' : P \prec Q) \vee (\forall\ w \in \mathbb{R}^d_{\geq 0}\, \exists P \in \mathcal{S}' : w^T p < w^T q.$$

The maximal set of linearly non-dominated paths (i.e. for all $\hat{\mathcal{S}} \supsetneq \mathcal{S}'$ exists $Q \in \hat{\mathcal{S}} \setminus \mathcal{S}'$ such that $\mathcal{S}' \prec_{lin} Q$) is referred to as linear skyline.

In contrast to conventional domination, testing for linear domination might require comparing an object to more than one other object. That is, unless an object is already conventionally dominated. The definition also implies that the set of linearly non-dominated paths is a subset of the conventionally non-dominated paths. Figure 1(a) and 1(b) illustrate both concepts on the same two-dimensional dataset.

The term linear dominance has a graphical intuition: Consider a two-dimensional cost space. One would expect K linearly non-dominated paths $\{P^1, \ldots, P^K\}$ to



(a) Check if p is linearly dominated by o, q

(b) Pruning area of paths

Fig. 2. Linear Domination areas

dominate another path Q if q lies "behind" any line that can be spanned by the elements of $\{p^1, \ldots, p^K\}$. Thus, the area being linearly dominated by $\{p^1, \ldots, p^K\}$ corresponds to the the part of the convex hull being directed towards the origin. Furthermore, the line connecting q to the origin has to intersect this hull.

We show that this intuition coincides with our definition for two-dimensional cost spaces. The proof for an arbitrary number of dimensions is not straight forward and therefore omitted here.

Theorem 1 *Let O and Q be two paths with $o_1 < q_1$ and $q_2 < o_2$ and let P be another path, for which we want to know if it is linearly dominated by $\{O, Q\}$. Let n be the normal vector of the line between O and Q, such that $n^T o = n^T q$. u shall be the component-wise minimum of o and q, such that $u_1 = min(o_1, q_1)$ and $u_2 = min(o_2, q_2)$. Then, it follows that:*

$$\{o, q\} \prec_{lin} p \quad \Leftrightarrow \quad u \prec p \ \wedge n^T p > n^T o = n^T q$$

This is illustrated in Figure 2.

Proof. If P is conventionally dominated by O or Q, P has to lie above the line between o and q such that $n^T p > n^T o = n^T q$ and p is also dominated by the component-wise minimum u of o and q. If p is not dominated by u, p has either a smaller x-value than both o and q, a smaller y-value than both, or dominates o or q. If p is not dominated by u, it is therefore not linearly dominated by $\{o, q\}$. In the remaining cases, p is not dominated by o or q but is dominated by u. From the statement $\{o, q\} \prec_{\text{lin}} p$ then follows that $\forall\, w \in \mathbb{R}_{\geq 0} : w^T o < w^T p \vee w^T q < w^T p$. This contradicts $n^T o = n^T q \geq n^T p$ for $w = n$ and therefore requires $n^T p > n^T o$. If p is dominated by u, p therefore lies above the line between o and q if it is linearly dominated by $\{o, q\}$. Accordingly if $n^T p \leq n^T o = n^T q$ there is a contradiction with $\{o, q\} \prec_{\text{lin}} p$. If p is dominated by u, p is therefore not linearly dominated by $\{o, q\}$ if it lies below the line or on the line between o and q.

Based on this comprehension of linear skylines, we now move on to introduce the algorithmic structure of our approach.

4 Linear Skyline Computation in Bicriterion Networks

In this section, we will describe our new method and its theoretical foundations. We will start with describing our solution to the most critical aspect of our algorithm, efficiently maintaining all linearly non-dominated paths between two nodes s and v. Afterwards, we will describe our complete algorithm and all its remaining steps in detail.

4.1 Managing Linear Skylines

As for conventional domination, it can be shown that for any linearly non-dominated path P from s to t all subpaths from s to any intermediate node $v \in P$ are non-dominated in the linear skyline $\mathcal{S}(s, v)$.

Lemma 1. *Let* $P = ((s, n_1)..,(n_i, v),..(n_j, t)) \in \mathcal{S}(s, t)$ *then any subpath* $Q = ((s, n_1),..,(n_i, v))$ *is linearly non-dominated in* $\mathcal{S}(s, v)$.

Proof. Given that the cost vector p of path P is linearly non-dominated, we know that $\exists\ w \in \mathbb{R}^d_{\geq 0}$ where $w^T p$ is optimal. If there would be a subpath $Q = ((s, n_1),..,(n_i, v))$ of P which is linearly dominated in $LS(s, v)$ then $w^T q$ cannot be optimal because that linear domination implies w-domination as well. Thus, exchanging Q with $\hat{Q}$ with $w^T \hat{q} < w^T q$ implies the existence of a path $\hat{P}$ with $w^T \hat{p} < w^T p$. This is a contradiction to $P \in \mathcal{S}(s, t)$.

Based on this lemma, it makes sense to only extend paths not being linearly dominated during traversal. Furthermore, after extending a path P from s to v, it has to be checked whether there are any other paths P^i, P^j linearly dominating P. If P is linearly non-dominated, it must be stored with v as part of its local linear skyline $\mathcal{S}(s, v)$. Furthermore, we have to delete any other $P^i \in \mathcal{S}(s, v)$ for which $\exists\ P^j \in \mathcal{S}(s, v) : \{p, p^j\} \prec_{\text{lin}} p^i$ holds.

This step is the most critical operation in our algorithm because it is performed frequently during each query. Moreover, the complexity increases with the number of elements in $\mathcal{S}(s, v)$. Thus, optimizing this operation is the key to efficiently computing linear skylines.

A naive solution to solve this problem is to compare the new path P to any pair $P^i, P^j \in \mathcal{S}(s, v), i \neq j$,. If a new path P is non-dominated, we also need to check whether for any pair $P^i, P^j \in \mathcal{S}(s, v)$, $\{p, p^j\} \prec_{\text{lin}} p^i$ holds. All paths P^i for which this property holds must be deleted from $\mathcal{S}(s, v)$. Both steps have a quadratic complexity in the number of elements K being contained in $\mathcal{S}(s, v)$.

A characteristic of linear skylines in a two-dimensional cost space is that sorting the values of the first criterion in ascending order implies an descending order in the second criterion. Formally, an ordering of $\mathcal{S}(s, v)$ can be expressed as a tuple $(P^1, \ldots, P^K)$ where $\forall\ i < j \in \{1,..,K\}$ the following conditions hold:

$p_1^i < p_1^j \wedge p_2^i > p_2^j$. This follows directly from the definition of conventional domination.

For a given path P and a linear skyline $\mathcal{S}(s,v)$, we refer to the elements $P^k, P^{k+1} \in \mathcal{S}(s,v)$ being closest to p w.r.t. the first cost criterion as neighbors. We refer to P^k with $p_1^k \leq p_1$ as left neighbor and denote P^{k+1} with $p_1^{k+1} > p_1$ as right neighbor. At least one of both neighbors must exist, unless $\mathcal{S}(s,v) = \emptyset$. For checking linear dominance, it is only necessary to compare to the potential neighbors of P in $\mathcal{S}$ which is shown in the following lemma.

Lemma 2. *Given a path $P = ((s,v), \ldots, (u,t))$ and the linear skyline $\mathcal{S}(s,v) = (P^1, .., P^K)$. Then the following statements hold:*
Case 1: If $p_1^1 > p_1$ then P is linearly non-dominated in $\mathcal{S}(s,t)$.
Case 2: If $p_1^K < p_1 \wedge p_2^K > p_2$ then P is linearly non-dominated.
Case 3: If $\exists\ k \in \{1, .., K-1\}$ with $p_1^k \leq p_1 \wedge p_1^{k+1} \geq p_1$ and $\exists O, Q \in \mathcal{S}(s,v)$ with $\{o,q\} \prec_{lin} p$ then $\{p^k, p^{k+1}\} \prec_{lin} p$ holds as well.

Proof. Case 1: $p_1 < p_1^1 \Rightarrow p_1 < p_1^i$ with $1 \leq i \leq K$. Thus, P w-dominates all P^i for $w = (1,0)$. Case 2: $p_2 < p_2^K \Rightarrow p_2 < p_2^i$ with $1 < i < K$. Thus, P w-dominates all P^i for $w = (0,1)$. If neither case 1 nor case 2 applies, P has two neighbors and case 3 applies. Case 3: There are only two valid orderings for the values $o_1, q_1, p_1^k, p_1, p^{k+1}$ where $\{o,q\} \prec_{\text{lin}} q$ and $p_1^k \leq p_1 \leq p_1^{k+1}$ can both hold.
Subcase 1: $o_1 \leq q_1 \leq p_1^k \leq p_1 \leq p_1^{k+1}$. From $q_1 \leq p_1 \wedge o_1 < p_1$ follows $q_2 < p_2$ Since $P^k \in \mathcal{S}(s,v) \wedge q_1 \leq p_1^k \Rightarrow q_2 > p_2^k$. Therefore, $p_1^k \leq p_1 \wedge p_2^k \leq p_2$ which implies $p^k \prec p \Rightarrow \{p^k, p^{k+1}\} \prec_{\text{lin}} p$.
Subcase 2: $o_1 \leq p_1^k \leq p_1 \leq p_1^{k+1} \leq q_1$ This case can be shown by contradiction. Assume $\{p^k, p^{k+1}\} \prec_{\text{lin}} p$ does not hold. Then either $\{o,q\} \prec_{\text{lin}} p^k$ or $\{o,q\} \prec_{\text{lin}} p^{k+1}$ or both would hold. This is a contradiction to $P^k, P^{k+1} \in \mathcal{S}(s,v)$.

Practically, this observation allows us to reduce the amount of domination checks to one single check for each insertion candidate. Furthermore, determining the neighbors of P in $\mathcal{S}(s,v)$ can be performed using a binary search w.r.t. the first criterion. Thus, the complexity of checking whether P has to be inserted into $\mathcal{S}(s,v)$ has a time complexity of $O(\log |\mathcal{S}(s,v)|)$.

As mentioned above, we have to check whether inserting P leads to the deletion of other elements from $\mathcal{S}(s,v)$. The general idea of how to efficiently determine elements which are now linearly dominated is the following: All dominated elements have to be either conventionally dominated by P or linearly dominated by P and some other element $P^i \in \mathcal{S}(s,v)$. To find all linearly dominated paths efficiently (rather than testing all pairs), we propose to start two linear searches. The first search beginning with the left neighbor and the second search beginning with the right neighbor of P. As was shown in Lemma 2 a path P^i is only dominated, if it is dominated by both of its neighbors. Thus, it is sufficient to check whether P^k is dominated by P and its other neighbor P^{k-1} when searching to the left side and P^{k+1} when searching to the right side. Let us note that the left neighbor P^{k-1} can already conventionally dominate a path P^k. Thus, we can improve the search in the left direction by checking this

simpler property before testing for linear dominance. If P^k indeed is dominated, there might be more dominated objects in the same search direction and we would have to repeat the domination check with the new neighbor. On the other hand, if P^k is not linearly dominated, we can stop the search in this direction. Formally, this is formulated in the following lemma:

Lemma 3. *Given a linear skyline $\mathcal{S}(s,v) = (P^1, .., P^K)$ being ordered w.r.t. the first cost criterion and a new non-dominated Path P. For any $P^k \in \mathcal{S}(s,v)$ with $p_1^k < p_1$ it holds that if $\{p, p^{k-1}\} \prec_{lin} p^k$ does not hold then $\nexists\ i \in \{1, .., k\} : \{p, p^{i-1}\} \prec_{lin} p^i$. Correspondingly, for any $P^k \in \mathcal{S}(s,v)$ with $p_1^k > p_1$ it holds that if $\{p, p^{k+1}\} \prec_{lin} p^k$ does not hold,then $\nexists\ i \in \{k, .., K\} : \{p, p^{i+1}\} \prec_{lin} p^i$.*

Proof. From Lemma 2 we know that if P^k is linearly dominated it has to be linearly dominated by its neighbors P^{k-1} and P^{k+1}. If neither P^{k-1} nor P^{k+1} have been deleted from $\mathcal{S}(s,v)$, the new path P cannot be a neighbor of P^k and since P^{k-1}, P^k, P^{k+1} were members in the original linear skyline $\mathcal{S}(s,v)$, P^k must remain linearly non-dominated.

To conclude, when determining the dominated elements of $\mathcal{S}(s,v)$, we have to repeatedly check the current neighbors of P. If a neighbor is not linearly dominated, we can terminate the search in this direction. Updating the linear skyline $\mathcal{S}(s,v)$ has a linear worst case complexity in $K = |\mathcal{S}(s,v)|$ because in the worst case we have to delete any former elements of $\mathcal{S}(s,v)$. Let us note that the results of the dominance checks are negative in the majority of cases. Typically, we only have to perform the domination test having a logarithmic time complexity.

4.2 Computing Bicriterion Linear Skylines

After describing the efficient management of linear skylines, we will now specify our complete algorithm for computing linear skylines in bicriteria networks. In general, our algorithm starts to construct linearly non-dominated paths beginning with the start node s by successively selecting nodes and expanding all linearly non-dominated paths being discovered so far. Furthermore, we mark paths which have been already expanded to avoid duplicate expansions.

Our method employs two pruning rules to exclude paths which cannot be extended into a linearly non-dominated result path. The first is that a path ending at node v has to be linearly non-dominated in $\mathcal{S}(s,v)$. To enforce this rule, we maintain a local linear skyline for each visited node v and delete all paths that are linearly dominated.

The second pruning method is based on the condition whether a path P might be extended into a linearly non-dominated path between s and t. In other words, any path $P = ((s, n_1), ..(n_i, v))$ having a cost vector being already linearly dominated in $\mathcal{S}(s,t)$ does not have to be extended because any extension from v to t would only add costs. Thus, the extension would be dominated in $\mathcal{S}(s,t)$ as well. This pruning method can be considerably improved by adding a lower

Algorithm 1 BLRSC

Require: start s and target t
Ensure: Linear Route Skyline between s and t

```
Compute Minimal costs to t using backward Dijkstra search
init queue Q of nodes with s
while ¬ Q.isEmpty do
  n := Q.pop
  for all Path P ∈ S(s,n) do
    if ¬P.isExtended then
      if (p+n^min) not linearly dominated in S(s,t) /*forward estimation*/ then
        for all Outgoing Link (n,m) of Node n do
          O := extend(P,(n,m))
          Search P^k, P^{k+1} for O in S(s,m) using binary search
          if ¬({p^k, p^{k+1}} ≺_lin o) then
            i = k // prune dominated elements left of o
            while i > 2 ∧ {p^{i-1}, o} ≺_lin p^i do
              Delete p^i from S(s,m)
              i = i − 1
            end while
            i = k + 1 // prune dominated elements right of o
            while i < (|S(s,m)| − 1) ∧ {o, p^{i+1}} ≺_lin p^i do
              Delete p^i in S(s,m)
              i = i + 1
            end while
            Q.insertOrUpdate(m)
          end if
        end for
      end if
      P.setExtended
    end if
  end for
end while
return S(s,t)
```

bound of the cost values for each cost criterion. In [2] a reference point embedding is used to estimate the lower bound costs. This method has two drawbacks. The first is that the quality of the cost approximation strongly depends on the selected reference points. The second drawback is that a reference point embedding requires offline precomputation which limits the usablitiy of this solution under dynamically changing cost values. Recently, [3] proposed to generate optimal forward estimations during query processing by performing two reverse single-source all-target Dijkstra searches from the target t to the start s. The method starts by determining the shortest path $SP^1 = ((s, n_1), .., (n_i, t)$ with respect to criterion 1 und thus, derives an upper bound w.r.t. criterion 2 sp_2^1. Afterwards, we compute the shortest path to $SP^2 = ((s, m_1), .., (m_j, t)$ w.r.t. criterion 2 and thus, receive an upper bound for the costs in criterion 1 sp_1^2. Now we continue the search w.r.t. criterion 1 until all nodes having a smaller

cost than sp_1^2 are found. Finally, the search w.r.t. criterion 2 is continued until all nodes having a smaller distance than sp_2^1 are retrieved. This way, each visited node v can be labeled with a cost vector (v_1^{min}, v_2^{min}) representing the costs of the shortest paths between v and t. Though this precomputation step seems to be rather expensive, it offers a tight lower bound for any intermediate node which extremely increases the pruning power of the global pruning criterion testing forward estimations against already retrieved paths. Since the step constructing all linearly non-dominated paths is multiple times more expensive than the backsearch determining the lower bound costs, using this step still yields a dramatic runtime improvement.

Algorithm 1 describes our method in pseudocode. In the first step, we perform the precomputation step by performing the reverse single-source all-target Dijkstra searches starting with the target node t. In the next step, the forward traversal is started. We maintain a priority queue of nodes being ordered w.r.t. the first attribute in ascending order. The queue is initialized by adding the start node s. For each visited node v, we store a linear skyline which is managed as an ordered list. The priority of a node always corresponds to the smallest cost value of any linearly non-dominated path which has not been extended yet. Already extended paths are kept in the skyline to use them to dominate other paths, but are marked to prevent multiple extensions. In the main loop the algorithm pops the top node v. For each unprocessed path $P \in \mathcal{S}(s, v,)$, we check linear dominance in the current result skyline $\mathcal{S}(s, t)$ by adding the lower bound costs from v to t to the cost vector p. If p is dominated, it is flagged as already extended. If the forward estimation of p is not linearly dominated yet, the path is extended with all outgoing edges of node v. Each new path $\hat{P}$ ending at node n is checked for linear dominance in the current skyline $\mathcal{S}(s, n)$. If $\hat{P}$ has to be inserted into $\mathcal{S}(s, n)$, all linearly dominated results are removed from $\mathcal{S}(s, n)$ by successively searching the left and the right neighbors until a linearly non-dominated path is found in each direction. Furthermore, the node n is either inserted into the priority queue or its position in the queue is checked for a potential improvement. The algorithm terminates, if the queue is empty meaning that there is no path left which can be potentially extended into a result path.

5 Evaluation

All experiments are performed on a dedicated machine with an 3.0 Ghz Intel Xeon 5160 processor and 32 GB RAM. The algorithms were implemented in Java 1.7 and do not make use of multiple cores. Each task is consecutively solved by all compared algorithms and the execution order is randomized for each task. If an algorithm is not able to solve a task in less than 60 seconds the computation is aborted and it is counted as a timeout. Three separate runs of all tasks are performed and the average of these three runs is used. This experimental setup is intended to ensure similar evaluation conditions, eliminate possible evaluation order effects and smooth out runtime discrepancies.

There are four different graphs on which routing tasks are performed:

Table 1. Overview of absolute runtime averages. In case of timeouts (>60s) the rounded number of timeouts is denoted next to the † symbol.

cost criteria	time & distance		time & crossings		rand.$_1$ & rand.$_2$	
graph	G_{munich}	G_{bavaria}	G_{munich}	G_{bavaria}	$G_{250\times250}$	$G_{50\times50\times50}$
# of tasks	2450	90	2450	90	4	8
BLRSC	0.2s	4s	0.36s	4.83s	17s	22.49s
Multi-A^*	0.21s	5.13s	0.37s	>8.47s †3	30.4s	>55.4s †4
ARSC	0.25s	>21.06s †18	0.4s	>11.12s †6	>60s †4	>60s †8
Multi-BD	0.61s	>33.05s †32	0.41s	>17.54s †13	30.85s	41.3s

1. G_{munich} is a OpenStreet Map (OSM) road network of the area around Munich which covers 6 992 km^2 including the neighboring city Augsburg and consists of 782 030 nodes and 1 595 261 edges. The 2450 routing tasks on this graph are between 50 city districts and municipalities in and around Munich.
2. G_{bavaria} is a OSM road network of Bavaria which covers 70 549 km^2 and consists of 4 044 556 nodes and 8 298 017 edges. The 90 routing tasks on this graph are between 10 major Bavarian cities.
3. $G_{250\times250}$ is a 2d grid network with 62 500 nodes and 249 000 edges. The 4 routing tasks on this graph are between opposite corners of the grid.
4. $G_{50\times50\times50}$ is a 3d grid network with 125 000 nodes and 735 000 links. The 8 routing tasks on this graph are between opposite corners of the lattice.

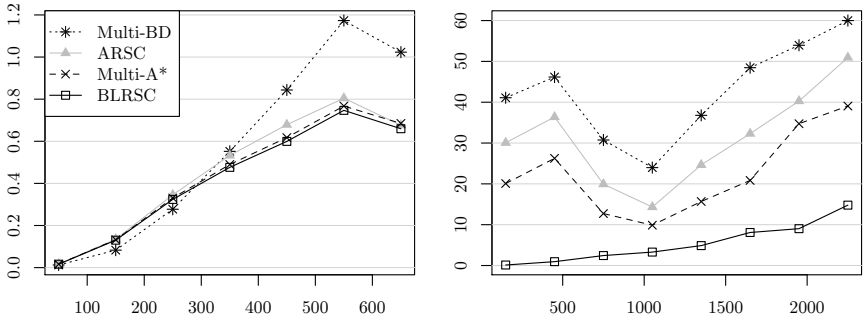
On G_{munich} and G_{bavaria}, the criteria time & distance and time & crossings are used for the routing tasks. On $G_{250\times250}$ and $G_{50\times50\times50}$, the criteria rand.$_1$ & rand.$_2$ are used which have independent pseudorandom cost values.

We compare our new method BLRSC to three other algorithms: The first is ARSC [2] to a have a comparision partner for computing conventional route skylines. We modified ARSC by using the same precomputation step proposed in [3] to achieve a better comparability to BLRSC. Multi-BD represents the state-of-the-art for computing supported solutions as proposed in [17]. For Multi-BD, the single objective shortest paths are computed using Bidirectional Dijkstra. However, the lower bounds in [3] can be used in combination with the approach from [17] as well which allows us to employ A^*-search instead of Bidirectional Dijkstra. We will refer to this algorithm as Multi-A^* and compare to this appraoch to evaluate how much of the performance gain is caused by the optimal lower bounds.

As can be seen in Table 1 the proposed algorithm BLRSC has the lowest runtime average in all experimental conditions, and it is the only algorithm in the experiments which solves all tasks in less than 60 seconds. As can be seen in Figure 3 this is the case throughout different task difficulties which is assessed through the number of hops between end points. The only exception can be seen in Figure 4(a), where Multi-BD is faster than BLRSC for less than 300 hops. BLRSC is also clearly faster for grid networks as can be seen in Figure 4. BLRSC is on average significantly faster than the previous approach Multi-BD. As can

Table 2. Overview of runtime averages relative to runtime averages of our main contribution BLRSC. In case of timeouts (>60s) the rounded number of timeouts is denoted next to the † symbol.

cost criteria	time & distance		time & crossings		rand.$_1$ & rand.$_2$	
graph	G_{munich}	G_{bavaria}	G_{munich}	G_{bavaria}	$G_{250\times250}$	$G_{50\times50\times50}$
# of tasks	2450	90	2450	90	4	8
BLRSC	1	1	1	1	1	1
Multi-A^*	1.05	1.28	1.02	>1.75 †3	1.78	>2.46 †4
ARSC	1.22	>5.25 †18	1.09	>2.29 †6	>3.52 †4	>2.66 †8
Multi-BD	2.98	>8.24 †32	1.12	>3.62 †13	1.81	1.83



(a) G_{munich} with time & crossings (b) G_{bavaria} with time & crossings

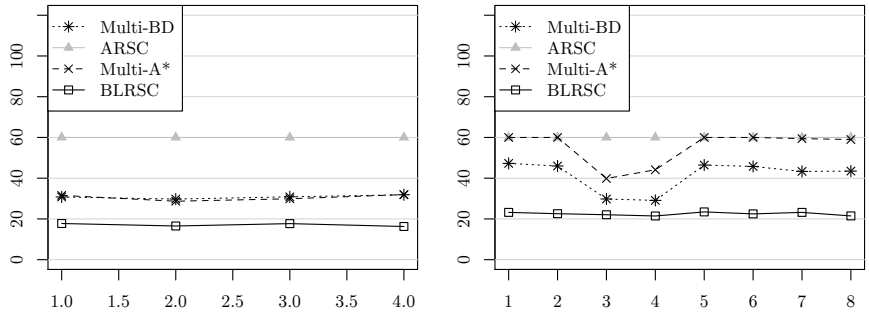
Fig. 3. Values on the x-axis are numbers of hops between end points and values on the y-axis are average runtimes in seconds.

be seen in Table 2, on G_{munich} with criteria time & distance it is about three times faster and on G_{bavaria} more than eight times faster, and on G_{bavaria} with criteria time & crossings it is more than three times faster.

ARSC performs very poorly on $G_{250\times250}$ and $G_{50\times50\times50}$, where it cannot solve a single task in less than 60 seconds which is also the worst performance out of all tested algorithms. ARSC is also more than five times slower for than BLRSC on G_{bavaria} with time & distance. Multi-A^* also performs poorly on $G_{250\times250}$ and $G_{50\times50\times50}$, where it is even slower than Multi-BD and much slower than BLRSC. It is also at least about twice as slow on G_{bavaria} using time & crossings than BLRSC. The experiments therefore show that BLRSC clearly outperforms all other tested approaches and that its performance gain is not only caused by the use the forward estimation. In other words, making use of the forward estimation with ARSC or using multiple A^* searches performs worse than BLRSC.

6 Conclusions

In this paper, we introduced linear skyline queries as a new query type in bicriteria network data. After discussing related work in different research communities,



(a) $G_{250\times250}$ with rand.$_1$ & rand.$_2$ (b) $G_{50\times50\times50}$ with rand.$_1$ & rand.$_2$

Fig. 4. Values on the x-axis are ids of the tasks and values on the y-axis are runtimes of three runs in seconds. If the value is 60s, it's a timeout and took longer than 60s.

we proposed a new algorithm which is based on efficient techniques for managing and updating linear skylines. Our new algorithm performs two steps. In a first precomputation step, we perform two network traversals to determine lower bounds for both cost criteria and all nodes being potentially visited during the search. In the next step, we use these lower bounds to perform a third graph traversal determining the linear skylines for all visited nodes until the linear skyline of the target is complete. In our experimental evaluation, we compare our new algorithm to the state-of-the-art algorithm for computing supported solutions which is the same task from a technical point of view. It can be observed that our new approach outperforms the compared methods, even though the compared methods were optimized by more efficient search techniques.

For future work, we would like to develop new efficient methods for determining linear path skylines in general multicriteria datasets. Furthermore, we would like to investigate linear path skyline computation under time-dependency, i.e. at least one cost criterion depends on the time an edge is visited.

References

1. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, Germany. (2001)
2. Kriegel, H.P., Renz, M., Schubert, M.: Route skyline queries: a multi-preference path planning approach. In: Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach, CA. (2010) 261–272
3. Machuca, E., Mandow, L.: Multiobjective route planning with precalculated heuristics. In: Proc. of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011). (2011) 98–107
4. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy. (2001)

5. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China. (2002)
6. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, CA. (2003)
7. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K.H., Zhang, Z.: On high dimensional skylines. In: Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany. (2006)
8. Zhang, Z., Guo, X., H.L., Tung, A.K.H., Wang, N.: Discovering strong skyline points in high dimensional spaces. In: CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management, Bremen, Germany. (2005)
9. LIN, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: the k most representitive skyline operator. In: Proceedings of the 23th International Conference on Data Engineering (ICDE), Istanbul, Turkey. (2007)
10. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: a semantic approach based on decisive subspaces. In: VLDB '05: Proceedings of the 31st international conference on Very large data bases,Trondheim, Norway. (2005)
11. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria. (2007)
12. Deng, K., Zhou, Y., Shen, H.T.: Multi-source query processing in road networks. In: Proceedings of the 23th International Conference on Data Engineering (ICDE), Istanbul, Turkey. (2007)
13. Huang, X., Jensen, C.S.: In-route skyline querying for location-based services. In: In Proc. of the Int. Workshop on Web and Wireless Geographical Information Systems (W2GIS), Goyang, Korea. (2004) 120–135
14. Jang, S.M., Yoo, J.S.: Processing continuous skyline queries in road networks. In: International Symposium onComputer Science and its Applications (CSA2008). (2008)
15. Ehrgott, M.: Multicriteria optimization. Springer (2005)
16. Hansen, P.: Bicriterion path problems. In: Multiple criteria decision making theory and application. Springer (1980) 109–127
17. Raith, A., Ehrgott, M.: A comparison of solution strategies for biobjective shortest path problems. Computers & Operations Research **36**(4) (2009) 1299–1331
18. Brumbaugh-Smith, J., Shier, D.: An empirical investigation of some bicriterion shortest path algorithms. European Journal of Operational Research **43**(2) (1989) 216–224
19. Guerriero, F., Musmanno, R.: Label correcting methods to solve multicriteria shortest path problems. Journal of Optimization Theory and Applications **111**(3) (2001) 589–613
20. Matthew Carlyle, W., Kevin Wood, R.: Near-shortest and k-shortest simple paths. Networks **46**(2) (2005) 98–109
21. Mote, J., Murthy, I., Olson, D.L.: A parametric approach to solving bicriterion shortest path problems. European Journal of Operational Research **53**(1) (1991) 81–92
22. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Algorithmics of Large and Complex Networks. (2009) 117–139
23. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerische Mathematik **1** (1959) 269–271

24. Nicholson, T.A.J.: Finding the shortest route between two points in a network. The computer journal **9**(3) (1966) 275–280
25. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on **4**(2) (1968) 100–107